Formal Proofs of Rounding Error Bounds

Pierre Roux

Noname manuscript No.
(will be inserted by the editor)

Formal Proofs of Rounding Error Bounds

With application to an automatic positive definiteness check.

Pierre Roux

Received: date / Accepted: date

Abstract Floating-point arithmetic is a very efficient solution to perform computations in the real field. However, it induces rounding errors making results computed in floating-point differ from what would be computed with reals. Although numerical analysis gives tools to bound such differences, the proofs involved can be painful, hence error prone. We thus investigate the ability of a proof assistant like Coq to mechanically check such proofs. We demonstrate two different results involving matrices, which are pervasive among numerical algorithms, and show that a large part of the development effort can be shared between them.

Keywords floating-point arithmetic · rounding error · numerical analysis · proof assistant · Coq · matrices · Cholesky decomposition

1 Introduction

Floating-point arithmetic is a very efficient solution to perform computations in the real field $\mathbb{R}$. Unfortunately, intermediate results of computations need to be rounded to fit in the floating-point format used. Due to this rounding errors, final results of computations differ from what would have been obtained by computing in the real field $\mathbb{R}$, although both results usually remain pretty close.

Fortunately, each rounding can only introduce a bounded error. By combining these atomic errors, one can get a bound on the error affecting the final result. Numerical analysis [10] thus aims at bounding these differences between results of numerical algorithms using floating-point or real arithmetic. Using such mathematical properties, rigorous results can be obtained despite the use of floating-point arithmetic [14], ensuring that no disastrous rounding error can happen during the computation.

This work was done while the author was a visiting researcher at LRI, Inria Saclay – Île-de-France.

Pierre Roux
ISAE, ONERA
E-mail: pierre.roux@onera.fr

More precisely, denoting f a function in the real field $\mathbb{R}$ and $\tilde{f}$ its actually computed floating-point counterpart, we have $\tilde{f}(x) = f(x) + e$. The value e is called the *forward error* and is expected to be negligible in front of $f(x)$. When $\tilde{f}(x) = f(x+d)$, d is called a *backward error*. The goal is then to prove some bound b, preferably as small as possible, such that $|e| \leq b(x)$ (respectively $|d| \leq b(x)$).

Proofs of this kind of mathematical results are hard to automate when they involve an arbitrary number of operations and are therefore mostly done by hand. However, they can be particularly painful and repetitive which make them specially error prone. That is why we want to investigate the ability of a proof assistant, namely Coq [1,6], to check them. Matrices being pervasive in numerical algorithms, we will particularly focus on them.

Formal proofs of error bounds have already been performed with proof assistants such as HOL [9] or Coq [3] or with automatic tools such as Gappa [7]. Yet, to the extent of author's knowledge, those work only address results with a fixed number of basic arithmetic operations whereas algorithms with an arbitrary, parameterized, number of operations are targeted in this paper.

Throughout the paper, `references` are made to our Coq development, available at `http://cavale.enseeiht.fr/formalbounds2014/`.

The paper is organized as follows. The remainder of this section first introduces our motivating example (Section 1.1) and a secondary example (Section 1.2), then gives basic properties of floating-point arithmetic (Section 1.3) and eventually details a simple proof about summations (Section 1.4). Section 2 then gives the detailed specification of floating-point arithmetic used while Section 3 shows how error terms can be combined. Section 4 eventually details proofs of numerical analysis results involving matrices, Section 5 deals with overflows, that were ignored until there, and Section 6 concludes.

1.1 Motivating Example: Cholesky Decomposition

We will use as motivating example throughout this paper a Cholesky decomposition which is a typical example of numerical algorithm involving matrices. Checking positive definiteness of matrices is one common use of Cholesky decomposition. A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is said to be positive semi-definite, written $A \succeq 0$, when for all $x \in \mathbb{R}^n$, $x^T A x \geq 0$.

To prove that a scalar $a \in \mathbb{R}$ is non negative, one can exhibit some $r \in \mathbb{R}$ such that $a = r^2$ (typically $r = \sqrt{a}$). Similarly, one can prove that a symmetric matrix $A \in \mathbb{R}^{n\times n}$ is positive semi-definite by exposing a matrix R such that $A = R^T R$ (for $x^T(R^T R)x = (Rx)^T(Rx) = \|Rx\|_2^2 \geq 0$ for all $x \in \mathbb{R}^n$). The Cholesky decomposition algorithm, shown on Figure 1, can compute such a matrix R, thus playing on matrices a similar role to the square root on reals. Indeed, taking as input a symmetric matrix $A \in \mathbb{R}^{n\times n}$, if the algorithm runs to completion, without ever attempting to take the square root of a negative value or perform a division by zero, it returns a matrix R such that $A = R^T R$ (this can be proved by induction on n). Thus, the mere termination of the Cholesky decomposition without error is enough to prove $A \succeq 0$. Conversely,

R **:=** 0;
for j **from** 0 **to** $n-1$ **do**
  for i **from** 0 **to** $j-1$ **do**
    $R_{i,j}$ **:=** $\left(A_{i,j} - \sum_{k=0}^{i-1} R_{k,i} R_{k,j}\right) / R_{i,i}$;
  od
  $R_{j,j}$ **:=** $\sqrt{A_{j,j} - \sum_{k=0}^{j-1} {R_{k,j}}^2}$;
od

Fig. 1 Cholesky decomposition: from a symmetric matrix $A \in \mathbb{R}^{n\times n}$, attempts to compute R such that $A = R^T R$. The resulting matrix R is upper-triangular (its elements $R_{i,j}$ with $i \leq j$ are the only one assigned after the initialization to 0). The elements $R_{i,j}$ are computed from left to right (outer loop) and from top to bottom (inner loop), each element being assigned exactly once a value obtained from the corresponding element $A_{i,j}$ of the input A and previously computed elements of R (i.e., elements on its top left). Due to the three nested loops (one being hidden in the Σ notation), the algorithm performs $\Theta(n^3)$ arithmetic operations (that is $n\sqrt{n}$ operations if n is the size of the input).

the decomposition will terminate without error for all symmetric positive definite matrix A (i.e., such that for all $x \in \mathbb{R}^n$, if $x \neq 0$ then $x^T A x > 0$).

This would work perfectly if the algorithm were run using real arithmetic. However, performing it with floating-point arithmetic, it could run to completion while $A \not\succeq 0$, due to rounding errors. That is, there could be $x \in \mathbb{R}^n$ such that $x^T A x < 0$. But rounding errors remain bounded, so that there exists a bound $c \in \mathbb{R}$, such that, if the floating-point Cholesky decomposition of A succeeds, then for all $x \in \mathbb{R}^n$, $x^T A x \geq -c \|x\|_2^2$. That is $A + cI \succeq 0$. The successful floating-point Cholesky decomposition of $A - cI$ eventually proves that $(A - cI) + cI = A \succeq 0$. Moreover, such a constant c can be easily computed from simple characteristics of A and the floating-point arithmetic format used[1].

It can be noticed that the method is sound but not complete. It may fail to prove that some matrices A are positive semi-definite. In particular, if A is positive semi-definite but not positive-definite (i.e., there is $x \neq 0$ such that $x^T A x = 0$), then $A - cI$ with $c > 0$ is not positive semi-definite ($x^T(A - cI)x < 0$ for the previous x) and the Cholesky decomposition will certainly fail. Thus, since the method only works for positive-definite matrices[2] $A \succ 0$, it could as well prove $A \succ 0$ rather than just $A \succeq 0$. This is done by choosing c such that if the decomposition succeeds, then for all x, $x^T A x > -c \|x\|_2^2$ (strict inequality instead of the wide one above). This ability to prove only strict, and not wide, inequalities is a common characteristic of this kind of numerical methods, due to the rounding errors [14].

[1] Indeed, replacing A by $A - cI$ leads to another bound c' which actually implies $(A - cI) + c'I \succeq 0$. However $c' \leq c$, hence $A \succeq 0$ (c.f., Corollary 4.4, page 15 for details).

[2] More generally, the method will fail to prove $A \succeq 0$ when the smallest eigenvalue of A is too small compared to c.

x **:=** 0;
while *true* **do**
  // u models input read from sensors
  u := ?(−1, 1); // random value in $\mathbb{R}^p$ with $\|u\|_\infty \leq 1$
  $x := Ax + Bu$;
  // send orders to actuators and wait for next period
od

Fig. 2 Model of a typical linear controller: such a controller can be used to maintain a physical system in a wanted state, when one can get some physical quantities of the system thanks to some sensors and has to act accordingly on the system through some actuators. The body of the infinite loop is repeatedly executed at periodic instants of time (for instance, every 10 milliseconds). At each iteration, the variable u first receives values from the sensors (this value being bounded, for instance between -1 and 1). Then, the variable x is updated according to its value from the previous iteration and the value of u. Finally, the value of x could be used to send some orders to the actuators.

1.2 Secondary Example: Ellipsoidal Invariants for Linear Controllers

Figure 2 shows a code modeling a typical linear controller. One may want to prove that, as long as the value of $u \in \mathbb{R}^p$ remains bounded ($\|u\|_\infty \leq 1$), the value of x in $\mathbb{R}^n$ will remain bounded along any execution of this code.

This is the case if and only if there exists a positive definite matrix $P \in \mathbb{R}^{n\times n}$ and a scalar $\lambda \in \mathbb{R}$ such that "$x^T P x \leq \lambda$" is an inductive loop invariant. That is "$x^T P x \leq \lambda$" holds before entering the loop (i.e., for $x = 0$, which amount to $\lambda \geq 0$) and if it holds before an iteration of the loop (i.e., if $x^T P x \leq \lambda$) then it still holds after the iteration (i.e., $(Ax+Bu)^T P (Ax+Bu) \leq \lambda$ for any u such that $\|u\|_\infty \leq 1$). Denoting $x = (x_0, \ldots, x_{n-1})$, the expression $x^T P x$ is a polynomial of degree 2 in the n variables $x_0, \ldots, x_{n-1}$ and when P is positive definite, the set $\left\{ x \in \mathbb{R}^n \;\middle|\; x^T P x \leq \lambda \right\}$ is, geometrically speaking, an ellipsoid. Since the variable x remains in this bounded set along any execution of the program, it remains bounded. All this is illustrated on Figure 3.

The goal will be to show that an invariant "$x^T P x \leq \lambda$" for an ideal program executed with real arithmetic (i.e., $Ax + Bu$ computed in the real field) remains valid, under some conditions, for the actual program using floating-point arithmetic (i.e., $Ax + Bu$ computed with floating-point arithmetic).

1.3 Definitions and Basic Properties

Definition 1.1 $\mathbb{F} \subset \mathbb{R}$ denotes the set of floating point values, $\circ : \mathbb{R} \to \mathbb{F}$ a rounding function (toward $+\infty$ or to nearest for instance) and $\mathrm{fl}(e) \in \mathbb{F}$ the floating point evaluation of expression e from left to right[3].

Example 1.2 Assuming 1, 2 and 3 are floating-point values, $\mathrm{fl}(1+2+3)$ denotes the value $\circ(\circ(1+2)+3)$.

[3] Order of evaluation matters since floating point operations are not associative.

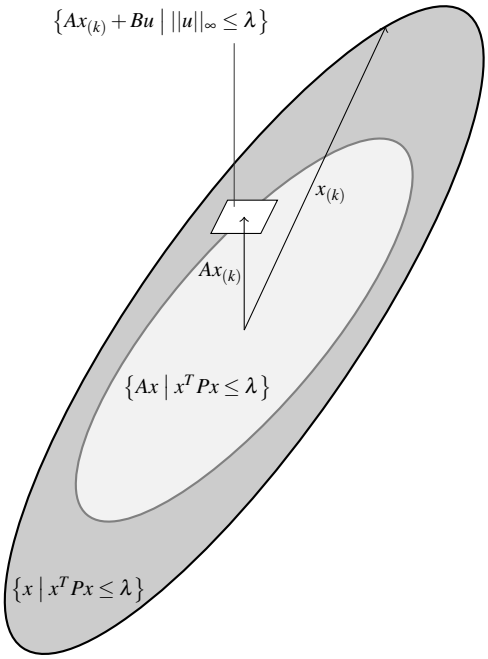

Fig. 3 "$x^T Px \leq \lambda$" is an inductive invariant: for any $x_{(k)}$ in the dark gray ellipse (i.e., $x_{(k)}^T P x_{(k)} \leq \lambda$), $Ax_{(k)}$ is in the light gray one. The white paralelogram represents the potential values of Bu for any bounded input u. Eventually, $Ax_{(k)} + Bu$ is in the original dark gray ellipse (i.e., $(Ax_{(k)} + Bu)^T P (Ax_{(k)} + Bu) \leq \lambda$). In geometric terms, the Minkowski sum of the light gray ellipse and the white paralelogram is included in the dark gray ellipse.

In practice, a floating-point value $f \in \mathbb{F}$ is encoded as a mantissa $m \in \mathbb{Z}$ and an exponent $e \in \mathbb{Z}$ such that $f = m\beta^e$ where β is the radix (commonly 2). Since f is encoded with a constant number of bits, m lies in a bounded range. More precisely, if the mantissa m is encoded with a precision of $prec$ bits: $|m| < \beta^{prec}$. To fully exploit the available precision, m and e can be chosen such that $|m| \geq \beta^{prec-1}$. Such numbers are called *normalized*.

The exponent e also lies in a bounded range. We will initially treat e as unbounded above which means that we ignore potential overflows[4]. In contrary, e is bounded below: $e \geq e_{\min}$. This leads to an underflow phenomena: numbers close to 0 (i.e., with absolute value less than $\beta^{e_{\min}+prec-1}$) cannot be represented with the usual precision of $prec$ bits. However, these numbers can be represented with a decreased precision when they are not too close from 0 (i.e., with absolute value larger than $\beta^{e_{\min}}$). Such numbers are called *denormalized*.

Thus, a real number $x \in \mathbb{R}$ not too close from 0 can be represented by a normalized number $\circ(x) \in \mathbb{F}$ such that $|x - \circ(x)| \leq \beta^{1-prec}|x|$ (or $\left(\beta^{1-prec}/2\right)|x|$ if $\circ$ is a

[4] They will be taken into account later (c.f., Section 5).

rounding to nearest). Otherwise, a denormalized number has to be used, and only $|x-\circ(x)| \leq \beta^{e_{min}}$ can be guaranteed (or $\beta^{e_{\min}}/2$ if $\circ$ is a rounding to nearest).

Definition 1.3 $\texttt{eps} \in \mathbb{R}$ and $\texttt{eta} \in \mathbb{R}$ are constants, depending on the floating-point format used, such that for all $x \in \mathbb{R}$, $\circ(x) \in \mathbb{F}$ satisfies either $|x-\circ(x)| \leq \texttt{eps}\,|x|$ or $|x-\circ(x)| \leq \texttt{eta}$.

Example 1.4 For the IEEE754 [11] binary64[5] format with $\circ$ a rounding to nearest, we have $\texttt{eps} = 2^{-53}$ ($\simeq 10^{-16}$) and $\texttt{eta} = 2^{-1075}$ ($\simeq 10^{-323}$).

These constants allow to bound the rounding errors of the basic arithmetic operations.

Property 1.5 For all $x, y \in \mathbb{F}$

$$\begin{aligned}
&\exists \delta \in \mathbb{R}, |\delta| \leq \texttt{eps} \wedge \mathrm{fl}(x \diamond y) = (1+\delta)(x \diamond y), && \text{for } \diamond \in \{+,-\}\\
&\exists \delta, \eta \in \mathbb{R}, |\delta| \leq \texttt{eps} \wedge |\eta| \leq \texttt{eta} \wedge \mathrm{fl}(x \diamond y) = (1+\delta)(x \diamond y) + \eta, && \text{for } \diamond \in \{\times,/\}\\
&\exists \delta \in \mathbb{R}, |\delta| \leq \texttt{eps} \wedge \mathrm{fl}\left(\sqrt{x}\right) = (1+\delta)\sqrt{x}.
\end{aligned}$$

For multiplication and division, the above property is a direct consequence of Definition 1.3. For addition, subtraction and square root, it also takes into account that, under mild conditions on the floating-point format, these operations are exact in case of underflow.

1.4 Simple Example: the Sum

The previous bounds on rounding errors of basic operations can be combined to get bounds on the error of larger expressions, as for instance a summation in the following classic result [10, 14].

We first define γ_n that will be extensively used in the following.

Definition 1.6 For all $n \in \mathbb{N}$, $\gamma_n := \frac{n\,\texttt{eps}}{1-n\,\texttt{eps}}$.

These terms are particularly useful to accumulate relative errors $(1+\delta)$ thanks, among others, to the following property.

Lemma 1.7 (`phi_gamma`) *For all $i, j \in \mathbb{N}$, $\delta \in \mathbb{R}^{j-i}$, if $2(j-i)\texttt{eps} < 1$ and for all k, $|\delta_k| \leq \texttt{eps}$, then*

$$\exists \theta,\ |\theta| \leq \gamma_{j-i} \wedge \prod_{k=i}^{j-1}(1+\delta_k) = 1+\theta.$$

Theorem 1.8 (`fsum_l2r_err_abs`) *For all $n \in \mathbb{N}$, $a \in \mathbb{F}^n$, if $2(n-1)\texttt{eps} < 1$, then*

$$\left| \mathrm{fl}\left(\sum_{i=0}^{n-1} a_i\right) - \sum_{i=0}^{n-1} a_i \right| \leq \gamma_{n-1} \sum_{i=0}^{n-1} |a_i|\,.$$

[5] Usual implementation of the type `double` in C.

The following proof is a good example of how equalities with error terms are used to derive such results. It gives a first overview of the properties and lemmas required about these error terms.

Proof We have by direct application of Property 1.5

$$\mathrm{fl}\left(\sum_{i=0}^{n-1} a_i\right) = \mathrm{fl}\left(\mathrm{fl}\left(\sum_{i=0}^{n-2} a_i\right) + a_{n-1}\right) = (1+\delta_n)\left(\mathrm{fl}\left(\sum_{i=0}^{n-2} a_i\right) + a_{n-1}\right)$$

for some $\delta_{n-1} \in \mathbb{R}$, $|\delta_{n-1}| \leq \mathtt{eps}$. Then

$$\mathrm{fl}\left(\sum_{i=0}^{n-1} a_i\right) = (1+\delta_{n-1})\left((1+\delta_{n-2})\left(\mathrm{fl}\left(\sum_{i=0}^{n-3} a_i\right) + a_{n-2}\right) + a_{n-1}\right)$$

for some $\delta_{n-2} \in \mathbb{R}$, $|\delta_{n-2}| \leq \mathtt{eps}$. By an immediate induction

$$\mathrm{fl}\left(\sum_{i=0}^{n-1} a_i\right) = \left(\prod_{j=1}^{n-1}(1+\delta_j)\right) a_0 + \sum_{i=1}^{n-1}\left(\prod_{j=i}^{n-1}(1+\delta_j)\right) a_i \tag{1}$$

for some $\delta_j \in \mathbb{R}$, $\left|\delta_j\right| \leq \mathtt{eps}$. Then by Lemma 1.7, for all $i \in [\![1, n-1]\!]$, there exist $\theta_{n-i} \in \mathbb{R}$ such that $|\theta_{n-i}| \leq \gamma_{n-i}$ and $\prod_{j=i}^{n-1}(1+\delta_j) = 1+\theta_{n-i}$, hence

$$\mathrm{fl}\left(\sum_{i=0}^{n-1} a_i\right) = (1+\theta_{n-1})\, a_0 + \sum_{i=1}^{n-1}(1+\theta_{n-i})\, a_i = \sum_{i=0}^{n-1} a_i + \left(\theta_{n-1}\, a_0 + \sum_{i=1}^{n-1}\theta_{n-i}\, a_i\right). \tag{2}$$

Finally, since for all $i \in [\![1, n-1]\!]$, $|\theta_{n-i}| \leq \gamma_{n-i} \leq \gamma_{n-1}$, there exists $\theta \in \mathbb{R}$ such that $|\theta| \leq \gamma_{n-1}$ and

$$\mathrm{fl}\left(\sum_{i=0}^{n-1} a_i\right) = \sum_{i=0}^{n-1} a_i + \theta \sum_{i=0}^{n-1} |a_i| \tag{3}$$

which enables to conclude.

One may notice that this result admits somewhat simpler proofs by direct induction. This proof was chosen in order to illustrate some basic use of the error terms δ and θ which play a key role in proofs of more complicated results[6].

It is also worth noting that this is a rather simple result and that proofs of more complicated results can rapidly involve more error terms. The use of a proof assistant then becomes a good way to get some confidence that no term is accidentally dropped or mixed with another which can easily happen with pen and paper proofs.

[6] Moreover, this choice improves modularity of the Coq code by sharing lemmas with other results.

2 Specification of Floating-Point Arithmetic

We now detail the formal specification of floating-point arithmetic used in our Coq development and already informally introduced in Section 1.3.

As seen in Property 1.5 and Theorem 1.8, both definitions and proofs make intensive use of real numbers with a bounded absolute value. To ease the manipulation of these error terms, we use a dependent record `bounded r` packing a real number with a proof that its absolute value is less than a non-negative real number `r`:

```
Record bounded (r : ℝ) :=
{ bounded_val :> ℝ; bounded_prop : |bounded_val| ≤ r }.
```

Since the set of floating-point values $\mathbb{F}$ is a subset of $\mathbb{R}$, we will similarly define floating point values as a value in $\mathbb{R}$ along with a proof that it lies in $\mathbb{F}$:

```
Record Ff format :=
{ F_val :> ℝ; F_prop : format F_val }.
```

where `format` is a predicate over $\mathbb{R}$ identifying real numbers that are in $\mathbb{F}$.

The floating-point arithmetic specification is then given by the following large record which will be used as parameter of all our subsequent developments.

```
Record Float_spec := {
(** format x means that x ∈ ℝ is a floating-point value *)
format : ℝ → Prop;
(** The type of floating-point values (coercible to ℝ). *)
𝔽 := Ff format;
(** 0 and 1 must be floating-point numbers. *)
format0 : format 0; format1 : format 1;
(** The opposite of a floating point number is a floating point number. *)
format_opp x : format x -> format (- x);
(** Bound on the relative error (normalized numbers, no underflow). *)
eps : ℝ; eps_pos : 0 ≤ eps; eps_lt_1 : eps < 1;
(** Bound on the absolute error (denormalized, when underflow occurs). *)
eta : ℝ; eta_pos : 0 < eta;
(** Some rounding. *)
frnd : ℝ → 𝔽; frnd_spec x :
∃ (d : bounded eps) (e : bounded eta), frnd x = (1+d)x+e :> ℝ;
(** Addition. *)
fplus (x y : 𝔽) : 𝔽 := frnd (x+y);
fplus_spec x y : ∃ d : bounded eps, fplus x y = (1+d)(x+y) :> ℝ;
fplus_spec2 (x y : 𝔽) : y ≤ 0 → fplus x y ≤ x;
(** Multiplication. *)
fmult (x y : 𝔽) : 𝔽 := frnd (x×y);
fmult_spec (x y : 𝔽) := frnd_spec (x×y);
fmult_spec2 x : 0 ≤ fmult x x;
```

```
(** Square root. *)
fsqrt (x : 𝔽) : 𝔽 := frnd √x;
fsqrt_spec x : ∃ d : bounded eps, fsqrt x = (1+d)√x :> ℝ;
}.
```

The unary minus `fopp` is then derived from `format_opp` and the subtraction and division of `x` and `y` are defined respectively as `fplus x (fopp y)` and `frnd (x/y)`.

Having performed our proofs with a proof assistant, we are guaranteed that the above record contains all the hypotheses about floating-point arithmetic used in these proofs. It is interesting to notice that this specification of floating-point arithmetic is really broad. In particular, it encompasses floating-point formats with gradual or abrupt underflow (i.e., denormalized numbers are respectively used or not) and any rounding mode. Fixed-point arithmetic can even be handled by just setting `eps` to 0, i.e., no relative, only absolute error occur. However, most of our developments are carried on with floating-point arithmetic in mind and the proved bounds might be pretty poor in a fixed-point arithmetic setting.

It is common practice in numerical analysis to ignore underflows [10]. Although this gives good indications on the numerical behavior of algorithms, underflows can appear with any practical implementation of floating-point arithmetic, potentially breaking such results. In our development, they are taken into account, thanks to the `eta` constant.

In our Coq development, the above specification of floating-point arithmetic is proved to hold for the floating point format with gradual underflow and rounding to nearest with any tie-break rule modeled in the Flocq library [4] with parameters corresponding to the binary64 format[5] (albeit without NaNs nor overflows, c.f., record `binary64` in our development). Other formats such as binary32[7] could be obtained by just modifying two constants defining size of the mantissa and minimal exponent.

In contrary to underflows, not handling NaNs and overflows does not constitute an actual issue. In fact, results considering those special values can easily be derived from results in our model with only finite values as will be shown in Section 5.

3 Combining Error Terms

3.1 Bounded Error Terms

Values of the type `bounded` defined at beginning of Section 2 are coercible to $\mathbb{R}$ and we developed a few helpful lemmas about them. The two following lemmas can be used to create such values.

Lemma 3.1 (`bounded_le_1`) $\forall r_1, r_2 \in \mathbb{R}, |r_1| \leq r_2 \Rightarrow \exists b : \texttt{bounded}\ 1,\ r_1 = b\, r_2$

Lemma 3.2 (`bounded_scale`)
$\forall r_1, r_2 \in \mathbb{R}, b_1 : \texttt{bounded}\ r_1, 0 < r_2 \Rightarrow \exists b_2 : \texttt{bounded}\ r_2,\ b_1 = b_2 \frac{r_1}{r_2}$

[7] Usual implementation of type `float` in C.

It is often needed to say that a value of type `bounded` b is also of type `bounded` b' for any $b' \geq b$ (for instance, in proof of Theorem 1.8, to state that the θ_{n-i} : `bounded` γ_{n-i} are all of type `bounded` γ_{n-1}):

Lemma 3.3 (`widen_bounded`)
$\forall r, r' \in \mathbb{R}, \forall b : \texttt{bounded}\ r, r \leq r' \Rightarrow \exists b' : \texttt{bounded}\ r',\ b = b'$

It is also common to get bounds of the form $b\,e$ with b : `bounded` r and e a complicated expression we want to replace by a simpler expression $e' \geq e$:

Lemma 3.4 (`bounded_larger_factor`)
$\forall r, r_1, r_2 \in \mathbb{R}, \forall b : \texttt{bounded}\ r, |r_1| \leq |r_2| \Rightarrow \exists b' : \texttt{bounded}\ r,\ b\,r_1 = b'\,r_2$

Error terms are compatible with basic arithmetic operations:

Lemma 3.5 (`bounded_opp`, `bounded_plus`, `bounded_mult`)

$\forall r : \mathbb{R}, \forall b : \texttt{bounded}\ r, \exists b' : \texttt{bounded}\ r,\ b' = -b$
$\forall r_1, r_2 : \mathbb{R}, \forall b_1 : \texttt{bounded}\ r_1, \forall b_2 : \texttt{bounded}\ r_2, \exists b' : \texttt{bounded}\ (r_1 + r_2),\ b' = b_1 + b_2$
$\forall r_1, r_2 : \mathbb{R}, \forall b_1 : \texttt{bounded}\ r_1, \forall b_2 : \texttt{bounded}\ r_2, \exists b' : \texttt{bounded}\ (r_1\,r_2),\ b' = b_1\,b_2$

Finally, probably the most important lemma about error terms allows to factor them and was exemplified between (2) and (3) in the proof of Theorem 1.8:

Lemma 3.6 (`bounded_distrl`, `big_bounded_distrl`)

$$\forall r, r_1, r_2 \in \mathbb{R}, \forall b_1, b_2 : \texttt{bounded}\ r, \exists b' : \texttt{bounded}\ r,\ b_1\,r_1 + b_2\,r_2 = b'\,(|r_1| + |r_2|)$$

$$\forall r, \in \mathbb{R}, \forall n : \mathbb{N}, \forall a \in \mathbb{R}^n, \forall b : (\texttt{bounded}\ r)^n, \exists b' : \texttt{bounded}\ r,\ \sum_i b_i\,a_i = b' \left(\sum_i |a_i| \right)$$

It is worth noting that this last property involves tuples (a and b) and sums of an arbitrary number of terms ($\sum_i b_i a_i$ for instance). Those are efficiently handled thanks to the `bigop` operator from the SSReflect library [2].

3.2 Accumulating Relative Errors

As already seen, for instance in the proof of Theorem 1.8 (between (1) and (2)), error terms of the form $(1+\delta_1)\dots(1+\delta_n)$, with $|\delta_i| \leq \texttt{eps}$, easily occur when relative errors accumulate. The terms $\gamma_n := \frac{n\,\texttt{eps}}{1-n\,\texttt{eps}}$ nicely enable to compact them into $1+\theta_n$, $|\theta_n| \leq \gamma_n$ as will be exposed in this section.

Most of the following lemmas require hypothesis of the form $n\,\texttt{eps} < 1$ for various values of n. In our Coq code, a set of small lemmas[8] allow to easily manipulate theses hypothesis so that they do not constitute an annoying burden in practice. First, a few very basic properties of the γ_n are proved. Namely, that they are non negative (provided $n\,\texttt{eps} < 1$), strictly less than 1 (provided $2n\,\texttt{eps} < 1$) and constitute a monotone sequence: for all $n \leq m$, $\gamma_n \leq \gamma_m$ (provided $m\,\texttt{eps} < 1$).

We then get some more interesting properties.

[8] For instance: $(n+1)\texttt{eps} < 1 \Rightarrow n\,\texttt{eps} < 1$.

Lemma 3.7 ([10, Lemma 3.3], `gamma_{mult{,_nat},plus{_mult,,_eps}`)
For all $n, m \in \mathbb{N}$

$$n \leq m \Rightarrow 2m\,\mathtt{eps} < 1 \Rightarrow \gamma_n\,\gamma_m \leq \gamma_n$$
$$(nm)\mathtt{eps} < 1 \Rightarrow n\,\gamma_m \leq \gamma_{nm}$$
$$(n+m)\mathtt{eps} < 1 \Rightarrow \gamma_n + \gamma_m + \gamma_n\,\gamma_m \leq \gamma_{n+m}$$
$$(n+m)\mathtt{eps} < 1 \Rightarrow \gamma_n + \gamma_m \leq \gamma_{n+m}$$
$$(n+1)\mathtt{eps} < 1 \Rightarrow \gamma_n + \mathtt{eps} \leq \gamma_{n+1}$$

Allowing to prove properties about the $\theta_n : \mathtt{bounded}\ \gamma_n$.

Lemma 3.8 ([10, Lemma 3.3], `gammap1_{mult{,_epsp1},div{_le,}}`)
For all $n, m \in \mathbb{N}$, *for all* $\theta_n : \mathtt{bounded}\ \gamma_n$, $\theta_m : \mathtt{bounded}\ \gamma_m$ *and* $\delta : \mathtt{bounded\ eps}$

$$2(n+m)\mathtt{eps} < 1 \Rightarrow$$
$$\exists \theta_{n+m} : \mathtt{bounded}\ \gamma_{n+m},\ (1+\theta_n)(1+\theta_m) = 1+\theta_{n+m}$$
$$2(n+1)\mathtt{eps} < 1 \Rightarrow$$
$$\exists \theta_{n+1} : \mathtt{bounded}\ \gamma_{n+1},\ (1+\theta_n)(1+\delta) = 1+\theta_{n+1}$$
$$m \leq n \Rightarrow 2(n+m)\mathtt{eps} < 1 \Rightarrow$$
$$\exists \theta_{n+m} : \mathtt{bounded}\ \gamma_{n+m},\ (1+\theta_n)/(1+\theta_m) = 1+\theta_{n+m}$$
$$2(n+2m)\mathtt{eps} < 1 \Rightarrow$$
$$\exists \theta_{n+2m} : \mathtt{bounded}\ \gamma_{n+2m},\ (1+\theta_n)/(1+\theta_m) = 1+\theta_{n+2m}$$

Lemma 3.9 (`phi_gamma,inv_phi_gamma`)
For all $i, j, n \in \mathbb{N}$, $\delta : (\mathtt{bounded\ eps})^n$, *if* $0 \leq i \leq j \leq n$ *then*

$$2(j-i)\mathtt{eps} < 1 \Rightarrow \exists \theta_{j-i} : \mathtt{bounded}\ \gamma_{j-i},\ \prod_{k=i}^{j-1}(1+\delta_k) = 1+\theta_{j-i}$$
$$(j-i)\mathtt{eps} < 1 \Rightarrow \exists \theta_{j-i} : \mathtt{bounded}\ \gamma_{j-i},\ \frac{1}{\prod_{k=i}^{j-1}(1+\delta_k)} = 1+\theta_{j-i}$$

It should be noted that, unlike the terms γ_n, the subscripts n in the notation θ_n are only present to improve readability but don't hold any semantic value.

It is also interesting to notice about the division [10, §3.4] that, although (according to Lemma 3.8) $(1+\theta_n)/(1+\theta_m) = 1+\theta_{n+m}$ only holds for $m \leq n$, according to the above lemma $\prod_n(1+\delta_i)/\prod_m(1+\delta_i) = 1+\theta_{n+m}$ even when $m > n$.

The notations δ and θ_n, with $|\delta| \leq \mathtt{eps}$ and $|\theta_n| \leq \gamma_n$, used in the above lemmas are particularly convenient and popular to carry proofs about error bounds. In fact, like the Landau big O notation, they greatly simplify proofs by enabling the use of simple equalities[9] instead of a collection of inequalities, or limits. However, it is easy to misuse them or forget hypotheses, such as $m \leq n$ in Lemma 3.8. The use of a proof assistant ensures that this does not happen.

[9] See for instance the proof of Theorem 1.8, page 6.

3.3 First Applications

Thanks to all the above lemmas, the Theorem 1.8, bounding the rounding error of a sum, is easily proved (c.f., `fsum_l2r_err_abs` in our Coq development). Similar results are also proved for the dotproduct of two vectors of floating point values.

Lemma 3.10 (`fdotprod_l2r_err_abs`)
For all $n \in \mathbb{N}$ and $a, b \in \mathbb{F}^n$, if $2n\,\texttt{eps} < 1$, then

$$\left| \mathrm{fl}\left(\sum_{i=0}^{n-1} a_i b_i \right) - \sum_{i=0}^{n-1} a_i b_i \right| \leq \gamma_n \left(\sum_{i=0}^{n-1} |a_i b_i| \right) + 2n\,\texttt{eta}.$$

Another example, in case the first operand a is constituted of real numbers, which have to be rounded to floating-point values prior to computation of the dotproduct. This will be used in the last application of this paper.

Lemma 3.11 (`fdotprod_l2r_fstr_err`)
For all $n \in \mathbb{N}$, $a \in \mathbb{R}^n$ and $b \in \mathbb{F}^n$, if $2(n+1)\texttt{eps} < 1$, then

$$\left| \mathrm{fl}\left(\sum_{i=0}^{n-1} a_i b_i \right) - \sum_{i=0}^{n-1} a_i b_i \right| \leq \gamma_{n+1} \left(\sum_{i=0}^{n-1} |a_i b_i| \right) + 2 \left(n + \sum_{i=0}^{n-1} |b_i| \right) \texttt{eta}.$$

Due to the presence of existential quantifiers in most intermediate lemmas, proof style in the Coq proof assistant heavily relies on forward proving. This does not appear to add much burden to the proof writing process, as long as proofs are well structured into lemmas of reasonable size[10]. Otherwise, one can first provide some dummy terms and later step back to replace them by the, then more obvious, adequate terms. Using the `evar` mechanism of Coq might also be a solution. In the "big enough numbers" for ε, η proofs about limits [5], constraints of the form $\eta \leq \eta_k$ are first accumulated as a list of the η_k and then simplified as $\eta \leq \min_k \eta_k$. Unfortunately, we are manipulating equalities of arithmetic expressions involving error terms rather than inequalities and we do not see such an easy simplification in this setting.

4 Errors on Matrix Operations

4.1 Real Numbers Matrices

As stated in the introduction (Section 1.1), we intend to prove numerical analysis results on algorithms involving matrices. In our Coq development, we borrow matrix algebra from the SSReflect library [8]. But we also need some results which are specific to matrices of real numbers. We therefore introduce some basic definitions and lemmas about pointwise orders and absolute values, dotproducts and quadratic norms.

First, the pointwise extensions of the order $\leq$ and $<$ as well as the absolute value $|.|$ are defined and a group of lemmas are proved about them. Most of these lemmas

[10] Which is just good programming practice.

are just lifting of the existing results on the real field $\mathbb{R}$: reflexivity and transitivity of the order $\leq$, compatibility of this order with matrix addition or scaling, triangular inequality of the absolute value ($\forall A, B \in \mathbb{R}^{n\times m}, |A+B| \leq |A| + |B|$) and so on.

In order to deal with quadratic norms, we first define *positive (semi-)definite* matrices. A matrix $P \in \mathbb{R}^{n\times n}$ is said to be positive semi-definite, written $P \succeq 0$, when for all $x \in \mathbb{R}^n$, $x^T P x \geq 0$ and it is said to be positive definite, written $P \succ 0$, when for all $x \neq 0$, $x^T P x > 0$. Thus for a symmetric ($P^T = P$) positive definite matrix P, we define the *dotproduct* of two vectors $x, y \in \mathbb{R}^n$ as $x^T P y$ and the quadratic norm $\|x\|_P$ as $\sqrt{x^T P x}$. The dotproduct is then proved to actually be a dotproduct (bilinear, symmetric, definite and non negative). It follows that the quadratic norm is definite non-negative and satisfies the scaling property ($\|\lambda x\|_P = |\lambda| \, \|x\|_P$) which eventually enables to prove two usual inequalities: the Cauchy-Schwarz inequality:

$$\forall x, y \in \mathbb{R}^n, \ \left|x^T P y\right| \leq \|x\|_P \, \|y\|_P$$

and the triangular inequality:

$$\forall x, y \in \mathbb{R}^n, \ \|x+y\|_P \leq \|x\|_P + \|y\|_P.$$

In the particular case when $P := I$, the quadratic norm will be written $\|.\|_2$ and a few additional properties are proved: $\forall x, y \in \mathbb{R}^n$,

$$\begin{aligned}
&|x| \leq |y| \Rightarrow \|x\|_2 \leq \|y\|_2 \\
&\| |x| \|_2 = \|x\|_2 \\
&\left\| [1 \ldots 1]^T \right\|_2 = \sqrt{n} \\
&\|x\|_1 \leq \sqrt{n} \|x\|_2
\end{aligned}$$

where $\|x\|_1 := \sum_i |x_i|$.

We eventually needed 110 lemmas. Thanks to the nice SSReflect matrices [8], they are proved using only 394 lines of tactics (hence an average of 3.6 lines of tactic per lemma, the longest proof being the Cauchy-Schwarz inequality with 36 lines of tactic).

4.2 Main Application: Cholesky Decomposition

As explained in Section 1.1, given a matrix A we want to check its positive definiteness, that is to prove $A \succ 0$. This will be done by proving that there exists a constant $c \in \mathbb{R}$ such that when the Cholesky decomposition (c.f., Figure 1) of $A - cI$, performed with floating-point arithmetic, runs to completion without error (square root of negative value or division by zero), then $A \succ 0$. We follow the proof in [13][11].

The first lemmas proved deal with the two "basic blocks" of the Cholesky decomposition: the assignments performed in the inner then the outer loop (c.f., Figure 1, page 3). The two following lemmas are proved with tools similar to the one required for Lemmas 3.10 and 3.11 about floating-point sums and dotproducts.

[11] Actually, part of it. We only consider matrices of real numbers whereas [13] also handles complex numbers. [13] also offers improved bounds for sparse matrices and a non-positive-definiteness check.

Lemma 4.1 ([13, Lemma 2.1], `lemma_2_1`**)** *For all $n \in \mathbb{N}$, $a,b \in \mathbb{F}^n$, $c,d \in \mathbb{F}$, if $d \neq 0$ and $2(n+1)\mathtt{eps} < 1$, then*

$$\left| c - \sum_i a_i b_i - d\,\tilde{y} \right| \leq \gamma_{n+1} \left(\sum_i |a_i b_i| + |d\,\tilde{y}| \right) + 2\mathtt{eta}\,(k+1+|d|)\,.$$

where $\tilde{y} := \mathrm{fl}\left(\frac{c-\sum_i a_i b_i}{d}\right)$.

Lemma 4.2 ([13, Lemma 2.2], `lemma_2_2_{1,2}`**)** *For all $n \in \mathbb{N}$, $a \in \mathbb{F}^n$, $c \in \mathbb{F}$, if $2(n+2)\mathtt{eps} < 1$ and $\mathrm{fl}\left(c - \sum_i a_i^2\right) \geq 0$, then*

$$\left| c - \sum_i a_i^2 - \tilde{y}^2 \right| < \gamma_{n+2} \left(\sum_i a_i^2 + \tilde{y}^2 \right) + 2\mathtt{eta}\,(k+1)\,.$$

and

$$\tilde{y}^2 + \sum_i a_i^2 \leq \frac{c + 2\mathtt{eta}\,k}{1-\gamma_{n+2}}$$

where $\tilde{y} := \mathrm{fl}\left(\sqrt{c - \sum_i a_i^2}\right)$.

Then, given two matrices $A, \tilde{R} \in \mathbb{F}^{n\times n}$ the proposition `cholesky_spec` $A\ \tilde{R}$ expresses that $\tilde{R}$ is the floating-point Cholesky factor of A and is defined as follows

$$\begin{aligned} &\forall i,j \in [\![1,n]\!], i<j \Rightarrow \tilde{R}_{i,j} = \mathrm{fl}\left(\frac{A_{i,j} - \sum_{k=1}^{i-1} \tilde{R}_{k,i}\tilde{R}_{k,j}}{\tilde{R}_{i,i}} \right) \\ &\wedge \forall j \in [\![1,n]\!], \tilde{R}_{j,j} = \mathrm{fl}\left(\sqrt{A_{j,j} - \sum_{k=1}^{j-1} \tilde{R}_{k,j}^2} \right). \end{aligned} \tag{4}$$

It is proved that, from a matrix A, the algorithm presented in Figure 1, page 3 computes a matrix $\tilde{R}$ satisfying `cholesky_spec` $A\ \tilde{R}$ (c.f., file `cholesky_prog.v`).

Eventually, the proposition `cholesky_success` $A\ \tilde{R}$ states that the floating-point Cholesky decomposition of A runs to completion without error (and returns $\tilde{R}$):

$$\mathtt{cholesky\_spec}\ A\ \tilde{R} \wedge \forall i \in [\![1,n]\!],\ \tilde{R}_{i,i} > 0.$$

The constraint $\tilde{R}_{i,i} > 0$ obviously prevents divisions by zero. The way it prevents taking square roots of negative values is a bit more subtle. According to the specification of $\mathrm{fl}\left(\sqrt{\cdot}\right)$ (c.f., `fsqrt_spec`, page 9), $\mathrm{fl}(\sqrt{x}) = (1+d)\sqrt{x}$ and $\sqrt{x}$ is defined as 0 for all $x \leq 0$ in the Coq standard library for reals. Thus, $\mathrm{fl}(\sqrt{x}) > 0$ guarantees that $x > 0$.

With this specification of the floating-point Cholesky decomposition, the following main theorem can be proved about it.

Theorem 4.3 ([13, Theorem 2.3]) *For all $n \in \mathbb{N}$ ($n \geq 1$), for all $A, \tilde{R} \in \mathbb{F}^{n\times n}$, $m \in \mathbb{R}$, if $2(n+2)\mathtt{eps} < 1$, $A^T = A$, for all i, $A_{i,i} \leq m$ and* `cholesky_success` $A\ \tilde{R}$*, then*

$$\forall x \in \mathbb{R}^n, x \neq 0 \Rightarrow -|x|^T\,\Delta_A\,|x| < x^T A x$$

where $\Delta_{A\,i,j} := \alpha_{i,j}\,d_i\,d_j + 4\mathtt{eta}\,(n+2+m)$ *with* $\alpha_{i,j} := \gamma_{\min(i,j)+2}$ *and* $d_i := \sqrt{\frac{A_{i,i}+2i\mathtt{eta}}{1-\alpha_{i,i}}}$.

This theorem is proved thanks to the above Lemmas 4.1 and 4.2 and the lemmas about matrices of real numbers described in Section 4.1.

Here is an idea of the kind of tiny mistakes that can be hard to spot in pen and paper proofs and may only be unveiled when trying to mechanically check them. In the original paper [13], the proof accidentally made the assumption that $\tilde{R}_{i,i} \leq A_{i,i}$ where $\tilde{R}_{i,i} := \mathrm{fl}\left(\sqrt{A_{i,i} - \sum R_{k,j}^2}\right)$. This may be slightly false since $\sqrt{x}$ can be larger than $x \in (0,1)$ but this was completely overlooked by the author of this paper[12] when first checking the proof by hand. Although very minor, the issue only appeared when translating the proof in Coq (and was easily fixed since $\sqrt{x} \leq x+1$).

The previous theorem is pretty useless by itself but the following corollary looks closer from what we are looking for.

Corollary 4.4 ([13, Corollary 2.4]) *For all $n \in \mathbb{N}$ ($n \geq 1$), for all $A, \tilde{A} \in \mathbb{F}^{n\times n}$, $c \in \mathbb{R}$, if $2(n+2)\mathtt{eps} < 1$, $A^T = A$ and for all i, $0 \leq A_{i,i}$ and*

$$\forall x \in \mathbb{R}^n,\ \|x\|_2 = 1 \Rightarrow |x|^T \Delta_A |x| \leq c$$

and $\tilde{A}^T = \tilde{A}$ and

$$\forall i, j \in [\![1,n]\!],\ i < j \Rightarrow \tilde{A}_{i,j} = A_{i,j}$$
$$\forall i \in [\![1,n]\!],\ \tilde{A}_{i,i} \leq A_{i,i} - c$$

then, if there exists $\tilde{R} \in \mathbb{F}^{n\times n}$ such that `cholesky_success` $\tilde{A}\ \tilde{R}$*, we have*

$$A \succ 0.$$

Proof (sketch) By definition of $\tilde{A}$, for any $x \in \mathbb{R}^n$, if $\|x\| = 1$, then $x^T A x \geq x^T \tilde{A} x + c$. From the theorem, we get $x^T \tilde{A} x > -|x|^T \Delta_{\tilde{A}} |x|$. Since $\Delta_{\tilde{A}} \leq \Delta_A$ (elementwise inequality, c.f., `Delta_At_le_Delta_A` in the code), $x^T A x > -|x|^T \Delta_A |x| + c \geq 0$.

Finally, it is proved that any value larger than

$$\frac{\gamma_{2n+2}}{2}\mathrm{tr}(A) + 4\mathtt{eta}(n+1)\left(2(n+2) + \max_i A_{i,i}\right) \tag{5}$$

will work as constant c in the above corollary as long as $4(n+2)\mathtt{eps} < 1$. Thus, an appropriate constant c can easily be computed, for instance by over-approximating (5) with floating-point arithmetic with rounding toward $+\infty$. Then $\tilde{A}$ is computed by subtracting cI to A and the floating point Cholesky decomposition is performed. If it runs to completion without error, this rigorously proves that $A \succ 0$. This automatic positive definiteness check is efficient as it is performed with $\mathrm{O}(n^3)$ floating-point operations for a matrix A of size $n \times n$.

The bound $4(n+2)\mathtt{eps} < 1$ is not a practical issue since it only implies $n < \frac{1}{4\mathtt{eps}} - 2$ which is huge for practical values of `eps`. For instance, for the binary64 format, $\mathtt{eps} = 2^{-53}$ which leads to the constraint $n < 2^{51} - 2$ ($\simeq 10^{15}$). In practice, `eta` being very small compared to `eps` ($\mathtt{eps} = 2^{-53}$ and $\mathtt{eta} = 2^{-1075}$ for the binary64 format), the second term of the bound (5) is negligible with respect to the first term

[12] As well as the author and reviewers of the original paper.

$\frac{\gamma_{2n+2}}{2}\mathrm{tr}(A)$. This means that, when the diagonal coefficients of the matrix A are of order of magnitude[13] 1, the bound (5) is mostly $n^2\mathtt{eps}$.

Thanks to the previous corollary, positive definiteness check can be performed on matrices A of floating-point values ($A \in \mathbb{F}^{n\times n}$). However, if the matrix X we want to check has coefficients in the real field ($X \in \mathbb{R}^{n\times n}$), we first have to round them to floating-point values in $\mathbb{F}$ and we will end up checking some matrix $A \in \mathbb{F}^{n\times n}$ such that $|X-A|_{i,j} \leq R_{i,j} := \mathtt{eps}\,|A|_{i,j} + \mathtt{eta}$. Such interval matrices are easily handled thanks to the following corollary.

Corollary 4.5 ([13, Corollary 2.7]) *For all $n \in \mathbb{N}$ ($n \geq 1$), for all $A, \tilde{A} \in \mathbb{F}^{n\times n}$, $R \in \mathbb{R}^{n\times n}$, $c, r \in \mathbb{R}$, if $2(n+2)\mathtt{eps} < 1$, $A^T = A$, $R \geq 0$ and for all i, $0 \leq A_{i,i}$ and*

$$\forall x \in \mathbb{R}^n, \|x\|_2 = 1 \Rightarrow |x|^T \Delta_A\, |x| \leq c$$

and

$$\forall x \in \mathbb{R}^n, \|x\|_2 = 1 \Rightarrow |x|^T R\, |x| \leq r$$

and $\tilde{A}^T = \tilde{A}$ and

$$\forall i, j \in [\![1, n]\!], i < j \Rightarrow \tilde{A}_{i,j} = A_{i,j}$$
$$\forall i \in [\![1, n]\!], \tilde{A}_{i,i} \leq A_{i,i} - c - r$$

then, if there exists $\tilde{R} \in \mathbb{F}^{n\times n}$ such that `cholesky_success` $\tilde{A}\,\tilde{R}$*, we have*

$$\forall X \in \mathbb{R}^{n\times n}, X^T = X \Rightarrow |X-A| \leq R \Rightarrow X \succ 0.$$

Since $n \max\left\{R_{i,j} \;\middle|\; i, j \in [\![1, n]\!]\right\}$ is a suitable value for r, this gives an effective criterion for positive definiteness of a matrix X with coefficients in the real field $\mathbb{R}$.

The whole Coq development eventually counts 3.8 kloc. Among them, 0.4 are devoted to the specification of floating-point arithmetic (described in Section 2), 0.2 to bounded error terms (Section 3.1), 0.6 to the γ_n terms and their properties (Section 3.2), 0.4 to basic lemmas about sums and dotproducts (Sections 1.4 and 3.3) and 0.8 to matrices of real numbers (Section 4). Finally, the main theorem and corollaries (this section) take 1.2 kloc, and the remainder (0.2 kloc) is constituted of miscellaneous lemmas. This appears particularly reasonable, considering the original result is a far from trivial 6 page long paper proof [13].

4.3 Another Application: Impact of Rounding Errors on Ellipsoidal Invariants

To assert the reusability of our developments for numerical analysis results involving matrices, we targeted another property. We will see that three quarters of the previous development can be directly reused.

As introduced in Section 1.2, we are given an invariant "$x^T P x \leq \lambda$" for the program of Figure 2 executed with real arithmetic and we want to prove, under some

[13] A preprocessing can fit all diagonal coefficients in the interval $[0.25, 1]$. Indeed, $A \succ 0$ when $A' := D^T A D \succ 0$ for any non-singular matrix D. By choosing $D = \mathrm{diag}(d_0, \ldots, d_{n-1})$, we have $A'_{i,j} = d_i d_j A_{i,j}$. By choosing the d_i as powers of 2, these multiplications are exact in floating point arithmetic (if no underflow (nor overflow) occurs) and we can guarantee $A'_{i,i} \in [0, 25, 1]$ [13].

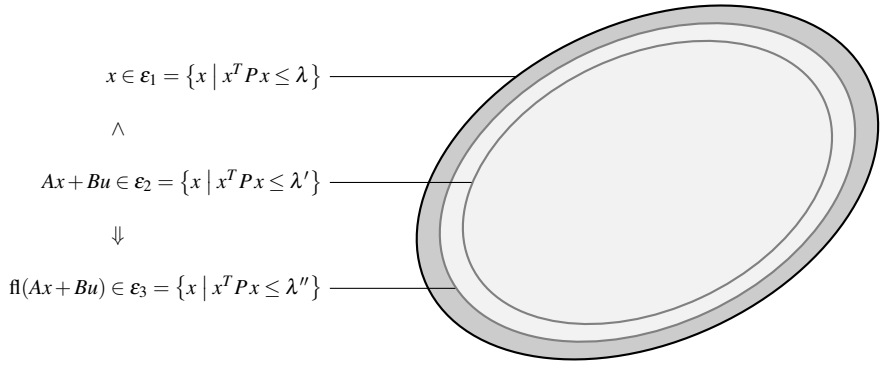

Fig. 4 For any x in the ellipsoid ε_1 of radius λ and assuming that $Ax+Bu$ is then in a smaller ellipsoid ε_2 of same shape and radius $\lambda' < \lambda$, it can be proved that $\mathrm{fl}(Ax+Bu)$ is in a slightly larger ellipsoid ε_3 of radius $\lambda'' > \lambda'$. The value of λ'' depends on λ, λ', A, B, P and the floating-point format. If $\lambda'' \leq \lambda$, this then proves that $\mathrm{fl}(Ax+Bu)$ remains in the original ellipsoid of radius λ. The figure is not to scale, the difference between λ' and λ'' being usually orders of magnitude smaller than with λ. It is also worth noting that it is enough to prove the inclusion $\{\mathrm{fl}(Ax+Bu) \mid x \in \varepsilon_1\} \subseteq \varepsilon_3$, set equality does not hold.

conditions, that it is also an invariant for the same program executed with floating-point arithmetic.

Here is the basic idea. The fact that the invariant is inductive for real arithmetic can be expressed as the following property

$$\forall x \in \mathbb{R}^n, \forall u \in \mathbb{R}^p, x^T P x \leq \lambda \Rightarrow \|u\|_\infty \leq 1 \Rightarrow (Ax+Bu)^T P (Ax+Bu) \leq \lambda.$$

But in practice, there will be some margin and we will be given the property

$$\forall x \in \mathbb{R}^n, \forall u \in \mathbb{R}^p, x^T P x \leq \lambda \Rightarrow \|u\|_\infty \leq 1 \Rightarrow (Ax+Bu)^T P (Ax+Bu) \leq \lambda'.$$

for some $\lambda' < \lambda$. That is if x is in the ellipsoid $\left\{x \mid x^T P x \leq \lambda\right\}$, then $Ax+Bu$ is in the smaller ellipsoid $\left\{x \mid x^T P x \leq \lambda'\right\}$. Computed with floating-point arithmetic, $\mathrm{fl}(Ax+Bu)$ will be equal to $Ax+Bu$ plus a small rounding error and will then lie in a slightly larger ellipsoid $\left\{x \mid x^T P x \leq \lambda''\right\}$ with $\lambda'' > \lambda'$. If $\lambda'' \leq \lambda$, then

$$\forall x \in \mathbb{R}^n, \forall u \in \mathbb{R}^p, x^T P x \leq \lambda \Rightarrow \|u\|_\infty \leq 1 \Rightarrow \mathrm{fl}(Ax+Bu)^T P \mathrm{fl}(Ax+Bu) \leq \lambda,$$

which means that the invariant is also inductive for floating-point arithmetic. This is illustrated on Figure 4.

The following theorem shows that $\lambda'' \leq \left(\sqrt{\lambda'} + \sqrt{\lambda}\, a + b\right)^2$ with a and b very small constants depending on A, B, P and the floating-point format used. That is λ'' is only slightly larger than λ' (and then hopefully less than λ).

Theorem 4.6 (`ellipsoid_error`) *For all $n, p \in \mathbb{N}$, $A, P \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times p}$, $s, s', \lambda, \lambda' \in \mathbb{R}$, if $2(n+p+1)\mathtt{eps} < 1$, $P^T = P$, $P \succ 0$, $sP - I \succeq 0$, $s'I - P \succeq 0$, then for all $x \in \mathbb{F}^n$, $u \in \mathbb{F}^p$ if*

$$x^T P x \leq \lambda,\ \|u\|_\infty \leq 1 \text{ and } (Ax+Bu)^T P (Ax+Bu) \leq \lambda'$$

then

$$\mathrm{fl}(Ax+Bu)^T P\,\mathrm{fl}(Ax+Bu) \le \left(\sqrt{\lambda'}+\sqrt{\lambda}\,a+b\right)^2$$

where $a := \gamma_{n+p+1}\sqrt{ss'}\sqrt{n}\|A\|_F + 2\sqrt{ss'}n\sqrt{n}\,\mathtt{eta}$
and $b := \gamma_{n+p+1}\sqrt{s'}\sqrt{p}\|B\|_F + 2\sqrt{s'}(n+2p)\sqrt{n}\,\mathtt{eta}$
where $\|M\|_F$ *denotes the Frobenius norm of the matrix* M *(i.e.,* $\|M\|_F := \sqrt{\sum_{i,j} M_{i,j}^2}$*).*

The value ss' is an upper bound on the condition number[14] of the matrix P. Intuitively, if ss' is close from 1, the ellipsoid $\left\{x \;\middle|\; x^T P x \le 1\right\}$ is close from a sphere. On the contrary, when the ellipsoid is much larger in some directions than other, ss' has to take larger values. Thus, the previous theorem gives better (i.e., smaller) bounds when the ellipsoid $\left\{x \;\middle|\; x^T P x \le 1\right\}$ is close from a sphere rather than very thin along some axes and very large along others.

For typical values ($n \le 10$, $p \le 10$, coefficients of A and B of order of magnitude 1, $ss' \le 10^4$, $s' \le \lambda$ and $\mathtt{eta} \ll \mathtt{eps} \simeq 10^{-16}$ for binary64), $\sqrt{\lambda}\,a+b \le 10^{-10}\sqrt{\lambda}$ which is very small with respect to a typical relative difference of 10^{-4} between λ and λ' [12]. Thus $\lambda'' \le \lambda$ always holds in practical cases.

Here are the main lines of the proof. $\mathrm{fl}(Ax+Bu)$ being defined as $\mathrm{fl}\left([A\ B][x^T\ u^T]^T\right)$ (recall that $\mathrm{fl}(expr)$ means $expr$ computed from left to right), for all i, $\mathrm{fl}(Ax+Bu)_i$ is a dotproduct and Lemma 3.11 gives for all i, $\mathrm{fl}(Ax+Bu)_i = (Ax+Bu)_i + e$ with $|e| \le b(x,u)$ (c.f., Lemma 3.11 for the details of b). We now have to bound x and u. $\|u\|_\infty \le 1$ is an hypothesis. Then, the hypothesis $sP - I \succeq 0$ and $x^T P x \le \lambda$ allow to prove $\|x\|_\infty \le \sqrt{s\lambda}$ (`lemma_2` in `ellipsoid_error.v`, proved using the lemmas on matrices of Section 4.1). These are the main ingredients of the following lemma which gives a bound on e only depending on λ and characteristics of A, B, P and the floating-point format.

Lemma 4.7 (`lemma_3` **in** `ellipsoid_error.v`) *For all* $n, p \in \mathbb{N}$, $A, P \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times p}$, $s, \lambda \in \mathbb{R}$, *if* $2(n+p+1)\mathtt{eps} < 1$, $P \succ 0$, $sP - I \succeq 0$, *then for all* $x \in \mathbb{F}^n$, $u \in \mathbb{F}^p$ *if*

$$x^T P x \le \lambda \text{ and } \|u\|_\infty \le 1$$

then there exists $d : (\mathtt{bounded}\ 1)^n$ *such that*

$$\mathrm{fl}(Ax+Bu) = Ax+Bu + \mathrm{diag}(d)\left(\sqrt{s\lambda}\,(\gamma_{n+p+1}\,c_A + 2n\,c_{\mathtt{eta}}) + \gamma_{n+p+1}\,c_B + 2(n+2p)c_{\mathtt{eta}}\right)$$

where $c_A := \left[\sum_j A_{0,j}, \ldots, \sum_j A_{n-1,j}\right]^T$, $c_B := \left[\sum_j A_{0,j}, \ldots, \sum_j A_{n-1,j}\right]^T$
and $c_{\mathtt{eta}} := [\mathtt{eta}, \ldots, \mathtt{eta}]^T$.

From the hypothesis $s'I - P \succeq 0$ of Theorem 4.6, we get for all $v \in \mathbb{R}^n$, $\|\mathrm{diag}(d)\,v\|_P \le \sqrt{s'}\|v\|_2$ (`lemma_4`) and the proof of the theorem finally follows thanks to matrix manipulations and lemmas of Section 4.1.

Among the 3.1 kloc needed to prove this theorem, 2.3 are shared with the development performed for the previous Section 4.2. Again, 0.8 kloc of Coq is a reasonably small amount of code for translating such a non trivial 4 page long paper proof.

[14] More precisely, $s'I - P \succeq 0$ implies that s' is an upper bound of all eigenvalues of P and $sP - I \succeq 0$ implies that $\frac{1}{s}$ is a lower bound of these eigenvalues.

5 Considering Potential Overflows

Until now, the possibility of overflows has been ignored. This section shows how previous results can be extended taking them into account. The basic idea is to use the fact that, as long as no overflow occurs, results are the same than in the previous model without overflows. An extension to overflows of the floating-point model of Section 2 is first given before demonstrating its use on our examples.

5.1 Specification of Floating-Point Arithmetic with Overflows

To the normalized and denormalized numbers already considered, the IEEE754 standard adds special values $-\infty$ and $+\infty$ to handle overflows and NaN (Not a Number) to handle undefined operations (for instance $0/0$). The new following specification handles these additional values by basically stating that when the result of an operation is not one of them, then the operation behaves as already defined in Section 2.

```
Record Float_infnan_spec := {
(** Type of floating-point values (either finite, infinite or NaN). *)
FI : Set; FI0 : FI;
(** finite f means that the floating-point number f is finite. *)
finite : FI → Prop; finite0 : finite FI0;
(** Underlying unbounded floating-point format. FI and F fis match when finite
holds. *)
fis : Float_spec;
(** Any float less than m (in absolute value) will be finite
(typically, m can be the smallest non representable positive float). *)
m : ℝ; m_ge_2 : 2 <= m;
(** Associates the corresponding value in F fis for finite values or 0 for infinities
and NaN. *)
FI2F : FI → F fis;
FI2F_spec x : (FI2F x ≠ 0 :> ℝ) → finite x;
FI2F0 : FI2F (FI0) = F0 fis :> ℝ;
(** Some rounding. *)
firnd : ℝ -> FI;
firnd_spec x : finite (firnd x) → FI2F (firnd x) = frnd fis x :> ℝ;
firnd_spec_f x : |frnd fis x| < m → finite (firnd x);
(** Opposite *)
fiopp : FI → FI;
fiopp_spec x : finite (fiopp x) → FI2F (fiopp x) = fopp (FI2F x) :> ℝ;
fiopp_spec_f1 x : finite (fiopp x) → finite x;
fiopp_spec_f x : finite x → finite (fiopp x);
(** Addition *)
fiplus : FI → FI → FI;
fiplus_spec x y : finite (fiplus x y) →
FI2F (fiplus x y) = fplus (FI2F x) (FI2F y) :> ℝ;
```

```
fiplus_spec_fl x y : finite (fiplus x y) → finite x;
fiplus_spec_fr x y : finite (fiplus x y) → finite y;
fiplus_spec_f x y : finite x → finite y →
|fplus (FI2F x) (FI2F y)| < m → finite (fiplus x y);
```
Multiplication and square root are very similar. Only the division slightly differs because $f/\pm\infty = 0$ for any finite f. They are all ommited fro brevity.
`}.`

It is worth noting that in Section 2, for $x, y \in \mathbb{F}$, $\mathrm{fl}(x+y)$ was defined as $\circ(x+y)$, using the addition $+$ over $\mathbb{R}$. This was made possible by the fact that $\mathbb{F}$ was a subset of $\mathbb{R}$. This is no longer the case as $\pm\infty$ and NaN are not in $\mathbb{R}$.

Another interesting subtlety to notice is that, if `finite` is satisfied by the largest representable float, the rounding of overflowing values forces the use of rounding to nearest. For instance, with rounding toward 0, an overflowing sum $x+y$ will result in the largest representable float (rather than $+\infty$ in rounding to nearest) and if it satisfies `finite`, this would violate `fiplus_spec`.

In our Coq development, the above specification is proved to hold for the bit-level model of binary64 in the Flocq library [4] (c.f., record `binary64_infnan`) with the `binary64` of Section 2 as underlying model `fis`.

5.2 Example: Cholesky Decomposition

This section proves that the absence of overflow during the execution of the Cholesky decomposition can be tested at runtime by simply checking that the diagonal coefficients of the computed matrix $\tilde{R}$ are finite (i.e., neither $\pm\infty$ nor NaN), in addition of the usual positivity check.

The specification `cholesky_spec_infnan` of the Cholesky decomposition is the same as (4), page 14, with the arithmetic operations replaced by the one of the new specification above. `cholesky_success_infnan` is then defined as

$$\texttt{cholesky\_spec\_infnan}\ A\ \tilde{R} \wedge \forall i \in [\![1,n]\!],\ \texttt{FI2F}\ \tilde{R}_{i,i} > 0.$$

It is worth noting that $\texttt{FI2F}\ \tilde{R}_{i,i} > 0$ implicitly states that $\tilde{R}_{i,i}$ is finite. This enables to prove the following lemma

Lemma 5.1 (`cholesky_success_infnan_cholesky_success`) *For* $A, \tilde{R} \in \texttt{FI}^{n\times n}$, *if* `cholesky_success_infnan` $A\ \tilde{R}$, *then* `cholesky_success` (`MFI2F` A) (`MFI2F` $\tilde{R}$) *with* `MFI2F` M *the matrix with coefficients* `FI2F` $M_{i,j}$.

Thus, if the other hypothesis of Corollary 4.4 hold, $\texttt{MFI2F}\ A \succ 0$ (i.e., $A \succ 0$ if all element of A are finite).

5.3 Example: Ellipsoidal Invariants

In Section 4.3, it was proved that a loop invariant $x^T P x \leq \lambda$ for the program of Figure 2 executed with real arithmetic still holds with floating-point arithmetic under

some conditions. Among these conditions, the positive definiteness $P \succ 0$ of the matrix P, ensures that the set $\left\{ x \mid x^T P x \leq \lambda \right\}$ is bounded. Thus, $\|x\|_\infty$ is bounded and if this bound is small enough, no overflow can occur.

This will be proved using the following lemma where flo(.) is the equivalent of fl(.) using the operation with overflow defined in Section 5.1.

Lemma 5.2 (`fdotprod_l2r_fstr_bounded`)
For all $n \in \mathbb{N}$, $a \in \mathbb{R}^n$ *and* $b \in \mathtt{FI}^n$, *if* $2(n+1)\mathtt{eps} < 1$, $2(n+1)\mathtt{eta} < 1$ *and*

$$\forall i, |a_i| < m' \wedge \mathtt{finite}\, b_i \wedge |b_i| < m'$$

with $m' := \sqrt{\frac{\mathtt{m}}{2(n+1)} - \mathtt{eta}} - \mathtt{eta}$, *then*

$$\mathtt{finite}\left(\mathrm{flo}\left(\sum_{i=0}^{n-1} a_i b_i\right)\right) \wedge \mathtt{FI2F}\left(\mathrm{flo}\left(\sum_{i=0}^{n-1} a_i b_i\right)\right) = \mathrm{fl}\left(\sum_{i=0}^{n-1} a_i \left(\mathtt{FI2F}\, b_i\right)\right).$$

This result is not surprising, it basically states that if all a_i and b_i are less than $\sqrt{\frac{\mathtt{m}}{n}}$, then no overflow happens during the computation of $\sum a_i b_i$ (since all $a_i b_i$ are less than $\frac{\mathtt{m}}{n}$ and the sum is then less than $\mathtt{m}$, the factor 2 and the `eta` in the definition of m' being only here to accomodate rounding errors (c.f., Lemma 3.11, page 12)).

According to Section 4.3, $\|x\|_\infty \leq \sqrt{s\lambda}$ when $sP - I \succeq 0$. Thus, no overflow can happen when $\sqrt{s\lambda} < m'$. This is proved in the next theorem.

Theorem 5.3 (`fiAxBu_bounded`) *For all* $n, p \in \mathbb{N}$, $A, P \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times p}$, $s, \lambda \in \mathbb{R}$, *if* $2(n+p+1)\mathtt{eps} < 1$, $2(n+p+1)\mathtt{eta} < 1$, $m' > 1$, $P \succ 0$, $sP - I \succeq 0$, $\sqrt{s\lambda} < m'$ *and for all* i, j, $\left|A_{i,j}\right| < m'$ *and* $\left|B_{i,j}\right| < m'$, *then for all* $x \in \mathbb{F}^n$, $u \in \mathbb{F}^p$ *if for all* i, `finite` x_i, `finite` u_i,

$$x^T P x \leq \lambda, \text{ and } \|u\|_\infty \leq 1$$

then

$$\mathtt{finite}\ (\mathrm{flo}(Ax+Bu)) \ \wedge \ \mathtt{MFI2F}\ (\mathrm{flo}(Ax+Bu)) = \mathrm{fl}(Ax+Bu)$$

where $m' := \sqrt{\frac{\mathtt{m}}{2(n+p+1)} - \mathtt{eta}} - \mathtt{eta}$.

Compared to Theorem 4.6, there are only a few additional hypothesis which are easily satisfied in practical cases:

- $2(n+p+1)\mathtt{eta} < 1$: for any reasonable floating-point format, the error `eta` for denormalized numbers is much smaller than `eps` for normalized numbers. This hypothesis is then subsumed by the previous $2(n+p+1)\mathtt{eps} < 1$.
- $m' > 1$: m' is usually very large, for instance for the binary64 format, $\mathtt{m} = 2^{1024}$, which means that $m' \geq 10^{146}$ since $n+p$ is bounded by $2(n+p+1)\mathtt{eps} < 1$.
- $\sqrt{s\lambda} < m'$: again, m' is large enough for this hypothesis to be satisfied in any practical case (typical values of $\sqrt{s\lambda}$ being less than 1000).
- $\left|A_{i,j}\right| < m'$ and $\left|B_{i,j}\right| < m'$: again easily satisfied (controllers are usually designed so that the coefficients of the matrices A and B remain somewhat close of order of magnitude 1, i.e., much smaller than m').

This section demonstrated our initial claim that overflows can be ignored in a first approach, since it is then possible to extend the results by either testing at runtime the absence of overflows or statically proving their absence.

The Coq development for this extension to overflows counts 1.1 kloc. Among them, 0.1 are devoted to the specification described in Section 5.1, 0.3 to prove that the bit-level model of binary64 from the Flocq library satisfy it, 0.2 to the Cholesky decomposition example and 0.5 to the ellipsoidal invariants.

6 Conclusion

We formally proved, using the proof assistant Coq [1,6], two results bounding rounding errors of numerical computations and involving matrices and common numerical analysis tools [10]. Our Coq development is available at `http://cavale.enseeiht.fr/formalbounds2014/`. It indicates that performing such proofs within proof assistants is tractable and that a large part of the proof effort could be reused for similar results. Our development is based on a broad high-level floating-point specification and we have proved that this specification is satisfied by the model of the binary64 format from the Flocq library of floating-point arithmetic for Coq [4].

The fact that we were able to translate, far from trivial, multiple pages paper proofs in about 1 kloc of Coq is a very encouraging achievement. It is also worth noting that performing mechanically checked proofs gave the opportunity to fix a few small mistakes in the proofs, thus asserting the interest of formalized proofs.

We eventually hope that a large part of our code can be reused in future developments, for instance about numerical integration of ODEs.

It would also be interesting to study a recent work offering sharper bounds [15].

Acknowledgements The author wants to express its deepest thanks to Sylvie Boldo and Guillaume Melquiond as well as to Érik Martin-Dorel and Pierre-Marie Pédrot for their help regarding this work.

References

1. Yves Bertot, Pierre Castéran, Gérard Huet, and Christine Paulin-Mohring. *Interactive theorem proving and program development : Coq'Art : the calculus of inductive constructions*. Texts in theoretical computer science. Springer, Berlin, New York, 2004. Données complémentaires http://coq.inria.fr.
2. Yves Bertot, Georges Gonthier, Sidi Ould Biha, and Ioana Pasca. Canonical big operators. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *TPHOLs*, volume 5170 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2008.
3. Sylvie Boldo, François Clément, Jean-Christophe Filliâtre, Micaela Mayero, Guillaume Melquiond, and Pierre Weis. Formal proof of a wave equation resolution scheme: The method error. In Matt Kaufmann and Lawrence C. Paulson, editors, *Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6172 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2010.
4. Sylvie Boldo and Guillaume Melquiond. Flocq: A Unified Library for Proving Floating-point Algorithms in Coq. In *Proceedings of the 20th IEEE Symposium on Computer Arithmetic*, pages 243–252, Tübingen, Germany, July 2011.
5. Cyril Cohen. Construction of real algebraic numbers in coq. In Lennart Beringer and Amy P. Felty, editors, *ITP*, volume 7406 of *Lecture Notes in Computer Science*, pages 67–82. Springer, 2012.
6. The Coq development team. *The Coq proof assistant reference manual*, 2012. Version 8.4.

7. Florent de Dinechin, Christoph Quirin Lauter, and Guillaume Melquiond. Assisted verification of elementary functions using Gappa. In Hisham Haddad, editor, *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC), Dijon, France, April 23-27, 2006*, pages 1318–1322. ACM, 2006.
8. Georges Gonthier, Assia Mahboubi, and Enrico Tassi. A Small Scale Reflection Extension for the Coq system. Research Report RR-6455, INRIA, 2008.
9. John Harrison. Floating point verification in HOL. In E. Thomas Schubert, Phillip J. Windley, and Jim Alves-Foss, editors, *Higher Order Logic Theorem Proving and Its Applications, 8th International Workshop, Aspen Grove, UT, USA, September 11-14, 1995, Proceedings*, volume 971 of *Lecture Notes in Computer Science*, pages 186–199. Springer, 1995.
10. Nicholas Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996.
11. IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. *IEEE Standard 754-2008*, 2008.
12. Pierre Roux and Pierre-Loïc Garoche. Computing quadratic invariants with min- and max-policy iterations: A practical comparison. In Cliff B. Jones, Pekka Pihlajasaari, and Jun Sun, editors, *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, volume 8442 of *Lecture Notes in Computer Science*, pages 563–578. Springer, 2014.
13. Siegfried Rump. Verification of positive definiteness. *BIT Numerical Mathematics*, 46:433–452, 2006.
14. Siegfried Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, May 2010.
15. Siegfried Rump and Claude Pierre Jeannerod. Improved backward error bounds for lu and cholesky factorizations. *SIAM Journal on Matrix Analysis and Applications*, 35(2):684–698, 2014.